JavaScript 参考 - JavaScript | MDN (mozilla.org)

Boolean

- 所有的返回false的情况：0.0、0、-0、空字符(不是空格)、false、NaN、null、undefined
- 空数组、空对象是true；

```
//new Boolean是true
var b = new Boolean(false);
    console.log(b);

    if (b) {
       console.log(`new Boolean(false) 是 true`);
  }
    console.log(b.valueOf()); //获取原始值
    //定义一个原始值 false  才是 false
    var bool = false;
    if (bool) {
        console.log('原始值false');
  }

  //转换成boolean类型
    console.log(Boolean(100));
    console.log(Boolean(0));
    console.log(Boolean(''));

  //使用!!把一个值转换成boolean类型console.log( ! !0);
    console.log(!!0.0);//false
    console.log(!!-0);//false
    console.log(!!''); //是空字符，不是空格console.log( ! !false); //false
    console.log(!!undefined);//false
    console.log(!!null);//false
    console.log(!!NaN);//false

  //转换为true:所有的对象都是
    trueconsole.log( !!{});//true
    console.log(!![]); //true
```

Undefined

- 只有一个值，是undefined；表示没有定义；
- 所有的返回undefined的情况：

- 声明的变量没有赋值；
- 访问数组不存在的下标；
- 访问对象不存在的属性；
- 函数没有返回值；
- 调用函数的时候没有给形参传值（函数内部声明的变量没赋值）；
- typeof一个没有声明的变量；

```
//声明的变量没有赋值
var a;
    console.log(a); //undefined

//访问数组不存在的下标
var arr = [1, 2, 3];
    console.log(arr[4]); //undefined

//访问对象不存在的属性
var obj = {
    a: 10,
    b: 20,
    c: 30
};
    console.log(obj.d); //undefined

//函数没有返回值
function fn() {}
    console.log(fn()); //undefined

//调用函数的时候没有给形参传值（函数内部声明的变量没赋值）
function sum(n, m) {
    console.log(n); //100
    console.log(m); //undefined
 }
sum(100);

//typeof一个没有声明的变量
     console.log(typeof abc); //undefined
```

1. null：object，万物皆空；

数组：Array

数组的创建

- new Array()：创建一个Array构造函数的实例化对象；

```
//数组的创建
  //1.通过数组字面量创建数组：
  var arr=[]; //创建一个空数组
  var arr1=[1,2,3,'8']; //创建一个有数组元素的数组，数组元素之间用逗号隔开
  //2.利用new Array( )创建数组：
  var arr2=new Array( ); //创建一个空数组
  var arr3=new Array(5); //创建一个长度为5的数组
  var arr4=new Array(1,2); //创建一个数组，数组内有两个元素，分别为1和2

//数组长度
  //数组长度通过 .length获取：
  console.log(arr4.length); //返回就是2，表示数组长度为2，对应有2个数组元素

//数组索引
  //数组内每一个元素对应着一个索引号，索引号默认从0开始（0,1，2，...）

  var arr5=[8,7,5,4,2,3]; //里面的第一个元素8对应的索引号就为0，元素7对应索引1，元素5对应索引3

//数组元素的获取
  //通过数组名[索引号]来获取：
  console.log(arr5[0]); //返回的就是8
  console.log(arr5[3]); //返回的就是4

//数组的遍历
 /**数组遍历，就是这个数组中有多少个元素，就让循环循环几次，
     //然后若有需要对每个数组元素做对应处理的程序，需要写在循环内部*/
var arr=[1,2,3,4,5,6];
 //遍历这个数组
for (var i=0; i<arr.length; i++){
    console.log(arr[i]);
}
 //因为数组索引号从0开始，所以循环也从0开始更加方便操作这个数组
```

- Array.from()：把一个可遍历的或类数组转换成真正的数组；

- Array.of()：注意和new Array的区别；
- fill(value, start, end)：用指定的元素对数组进行填充

```
<body>
  <ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
  </ul>
  <script>
  //如果要定义一个数组，我们就是用[]
  var arr1 = new Array(1, 2, 3);
      console.log(arr1);// [1,2,3]

  var arr2 = new Array(3, 4, 5);
      console.log(arr2); //[3,4,5]

  //Array.from()是一个静态方法
  var str = 'myname';
      console.log(Array.from(str)); //['m', 'y', 'n', 'a', 'm', 'e']

  //把类数组转换成真正的数组
  var list = document.querySelectorAll('li');
      console.log(list); //类数组
      console.log(Array.from(list)); //[li, li, li, li, li]

  //Array.of 是一个静态方法
      console.log(Array.of(6)); //这个数组只有一个元素 6
      console.log(new Array(6)); //只有一个数字，表示数组长度

  //数组的填充,fill()方法用一个固定值填充一个数组中从起始索引到终止索引内的全部元素。不包括终止索引
  var arr1 = new Array(3);
      arr1.fill(7);
      console.log(arr1);//[7,7,7]

  var arr3 = new Array(7);
```

```
      arr3.fill({
          name: '张三',
          age: 19
    }, 3, 6); //在3~6之间填充一个数组,不包含索引6
      console.log(arr3);
      //3: {name: '张三', age: 19}
      //4: {name: '张三', age: 19}
      //5: {name: '张三', age: 19}
  </script>
</body>
```

数组增删改

- pop()： 删除数组的最后一个元素并返回；
- push()： 在数组的尾部追加元素；
- shift()：删除数组的第一个元素并返回；
- unshift()：在数组的开始位置追加元素；
- splice()：可以在数组的任意位置进行增、删、改；
- slice(a,b)：返回由 a（包括）至 b（不包括）的元素所组成的数组，若一个参数都不传，则返回由全部元素组成的数组

```
var arr1 = [100, 200, 300];
    console.log(arr1); //[100, 200, 300]

//增 在数组的尾部追加新的元素
    arr1.push(400, 500, 600);
    console.log(arr1); //[100, 200, 300, 400, 500, 600]

//增 在数组的开始位置增加元素
    arr1.unshift('a', 'b', 'c');
    console.log(arr1); //['a', 'b', 'c', 100, 200, 300, 400, 500, 600]

//删除数组元素的方法
//pop( )方法，删除数组的最后一个元素，只能删除一个，与push( )对应，会改变原数组,括号内不跟参数
var arr1=[1,2,3,4,5]
    arr.pop( );
    console.log(arr1); //返回[0,1,2,3,4,5,]
//shift( )方法，删除数组的第一个元素，只能删除一个，与unshift( )对应，会改变原数组，括号内不跟参数
```

```
    arr1.shift( );
    console.log(arr1); //返回[1,2,3,4,5]

//改 下标修改
    arr1[0] = 'black';
    console.log(arr1); //['black', 'c', 100, 200, 300, 400, 500]

//splice(startIndex, length, value )方法，传入参数情况
 //对数组的任意位置的元素进行删除和修改、增加
 var arr2 = ['apple', 'pear', 'orange', 'banana'];
     arr2.splice(2, 1, '火龙果');
     console.log(arr2); //['apple', 'pear', '火龙果', 'banana']

 //删除或插入数组元素 splice( ) 方法，返回被删除元素组成的数组，会修改原数组
 //传入一个参数，表示索引号，自这个索引号开始一直到最后一个元素都被删除
 var arr=[1,2,3,4,5]
    console.log(arr.splice(3)); //返回的[4,5]
    console.log(arr); //返回[1,2,3],会修改原数组

 //传入两个参数，第一参数意义同上，第二个参数表示删除几个
    console.log(arr.splice(1,2)); //返回[2,3]
    console.log(arr); //返回[1]

 //传入两个以上的参数，前两个参数依然表示删除数组元素。
 //而后面的参数，则是要插入数组的元素，可以为任意数据类型，插入位置为第一个参数指定的位置
    console.log(arr.splice(1,0,2,3,4,5));//表示从索引为1的位置开始，删除0个元素，
 //并且从索引为1的位置，添加元素2,3,4,5;返回的是被删除的数组组成的元素，所以返回了一个空数组
    console.log(arr); //返回[1,2,3,4,5]
```

Array.slice和Array.splice的区别：

1 Array.slice方法： 接收两个参数：

- a：起始下标
- b：结束下标

返回由 a（包括）至 b（不包括）的元素所组成的数组，若一个参数都不传，则返回由全部元素组成的数组。

该方法执行不影响原数组元素

2 Array.splice方法： 接收若干参数：

- a：起始下标
- b：提取个数

- ...n：之后若干个是待插入的新元素

返回由从 a（包括）开始的 b 个元素组成的数组，并将 ...n 从 a 开始依次插入，若一个参数都不传，则返回空数组[]。

该方法的执行影响原数组元素。

数组查询

find()：返回数组中满足提供的测试函数的第一个元素的值

find内部的函数是对数组中的每个元素重复调用：

- item是元素；
- index是元素的索引；
- array是数组本身，可省略；

一般用于对对象数组的查询

如果返回true，则查询停止，返回item；如果没有查询到，则返回undefined

findIndex和find函数类似，只是返回的是元素的索引

但是这两个函数只是返回第一个查询为true的元素，因此只是一个值，若返回的是多个值，需要用filter

filter()： 方法创建一个新数组, 其包含通过所提供函数实现的测试的所有元素

```
//find()方法
 //找 长度>5的数组元素  短路机制
var arr3 = ['apple', 'pear', 'orange', 'banana', 'ABCDEFG'];
var findResult = arr3.find(function (el, ind, arr) {
    //el:当前遍历到的元素
    //index: 当前遍历到的索引，就是下标
    //arr:就是数组，用的少
     console.log(el, ind); //输出遍历到的元素和遍历到的索引（下标）
        //apple 0
        //pear 1
        //orange 2
     return el.length > 5
});
     console.log(findResult); //orange

//filter() 方法创建一个新数组，其包含通过所提供函数实现的测试的所有元素
var arr3 = ['apple', 'pear', 'orange', 'banana', 'ABCDEFG'];
var allRersult = arr3.filter(function (el) {
     return el.length > 5
```

```
});
    console.log(allRersult); //['orange', 'banana', 'ABCDEFG']
```

索引查询：

- findIndex
- includes
- indexOf
- lastIndexOf

```
//findIndex()方法返回数组中满足提供的测试函数的第一个元素的索引。若没有找到对应元素则返回-1
var array1 = [5, 12, 8, 130, 44];
var isLargeNumber = (element) => element > 13;
    console.log(array1.findIndex(isLargeNumber)); //  3

//includes() 方法用来判断一个数组是否包含一个指定的值，根据情况，如果包含则返回 true，否则返回 false
var arr = [1, 2, 3, 4];
    console.log(arr.includes(5)); //false

//indexOf()方法返回在数组中可以找到一个给定元素的第一个索引，如果不存在，则返回-1
var beasts = ['ant', 'bison', 'camel', 'duck', 'bison'];
    console.log(beasts.indexOf('bison'));  //  1
// start from index 2
    console.log(beasts.indexOf('bison', 2)); // 4
    console.log(beasts.indexOf('giraffe'));  //  -1


//根据元素找索引号
 //indexOf( )从前往后找
var arr=['pink','red','blue','purple','red'];
    console.log(arr.indexOf('red')); //返回1，即为找到的第一个复合条件元素索引号
    console.log(arr.indexOf('red',2)); // -1
    console.log(arr.indexOf('green')); //返回-1，因为原数组中没找到这个元素
 //lastIndexOf( ) 从后往前找
    console.log(arr.lastIndexOf('red')); //返回4
    console.log(arr.lastIndexOf('green')); //返回-1，因为原数组中没找到这个元素

//lastIndexOf() 方法返回指定元素（也即有效的 JavaScript 值或变量）在数组中的最后一个的索引，如果不存在则返回 -1。从数组的后面向前查找，从 fromIndex 处开始
```

```
var animals = ['Dodo', 'Tiger', 'Penguin', 'Dodo'];
    console.log(animals.lastIndexOf('Dodo')); //  3
    console.log(animals.lastIndexOf('Tiger'));  //  1
```

数组遍历

- forEach：遍历每一个元素

```
var arr = ['a', 'b', 'c'];
    arr.forEach(function(item, ind){
        console.log(item, ind);
    });

//forEach的使用:
<body>
    <ul>
        <li>夏装女</li>
        <li>夏装女</li>
        <li>夏装女</li>
    </ul>

    <script>
        /* 下面引入的js文件要调用一个方法__jp2，没有，怎么办？定义呗 */
        function __jp2(keyData){
            /** 初始化一个空白的html字符串*/
            var html ='';
            /** 获取到一个二维数组*/
            var dataArr = keyData.result;
            //使用forEach对数组进行循环操作
            dataArr.forEach(function(item){
                console.log(item);
                //html = html + `<li>${item[0]}</li>`;
                /* 在原来的基础之上，不断的追加新的li标签进来*/
                html += `<li>${item[0]}</li>`;
            });
            console.log(html);
            //把html标签显示到页面之上
            document.querySelector('ul').innerHTML = html;
        }
```

```
    </script>
   // <!-- 引入的文件只要内容是js代码就可以了，跟后缀没有关系 -->
    <script src="https://suggest.taobao.com/sug?k=1&area=c2c&q=%E5%A4%8F%E8%A3%85&code=utf-8&ts=1650452121346&callback=__jp2"></script>
</body>
```

- map（）创建一个新数组，这个新数组由原数组中的每个元素都调用一次提供的函数后的返回值组成

```
//map() 方法创建一个新数组，这个新数组由原数组中的每个元素都调用一次提供的函数后的返回值组成
<body>
    <ul>
        <li>1</li>
        <li>2</li>
        <li>3</li>
    </ul>

    <script>
        function jsonp1(keyData) {
            var data = keyData.result;
            var htmlarr = data.map(function (item) {
                return `<li>${item[0]}</li>`;
            });

            console.log(htmlarr);
            //把html标签显示在页面上
            //arr.join([separator])指定一个字符串来分隔数组的每个元素。如果需要，将分隔符转换为字符串。如果缺省该值，数组元素用逗号（,）分隔。如果separator是空字符串(""),则所有元素之间都没有任何字符。
            document.querySelector('ul').innerHTML = htmlarr.join('');
        };
    </script>
    <script
        src="https://suggest.taobao.com/sug?code=utf-8&q=%E5%A4%8F%E8%A3%85&extras=1&area=c2c&bucketid=atb_search&pid=mm_26632258_3504122_32538762&unid=&clk1=44c07cd701494145ea6aa32faaff2b05&_=1650453561329&callback=jsonp1">
    </script>
</body>
```

二者的区别：

- map方法会返回一个新的数组，数组元素是callback函数（map方法的参数）的返回值

- map有一个返回值，这个返回值是一个和原数组长度相等的新数组；{ }里什么也不写的话，所有元素就都是undefined；{ }写了return，并且return后面的值就是新数组的元素，直接更改这个return后面的值，就可以得到我们需要的新数组

遍历性能上来说 for循环遍历 < for...of遍历 < forEach遍历 < for...in遍历 < map遍历 :

```
// 1)经典的for循环:
   for(var i = 0; i < arr.length; i++){  }
// 2)for...in
   for(var i in  arr){  }
// 3) forEach
    arr.forEach(function(i){   })
// 4) map
    arr.map(function(i){   })
// 5) ES6的语法  for...of
    for(let i of arr){   }
```

数组其他方法

- concat( )方法用于多个数组的合并

```
var arr=[0,1,2,3,4,5]

 var newarr1=arr.concat(7,8);
    console.log(newarr1); //返回的是[0,1,2,3,4,5,6,7,8]
    console.log(arr); //返回的是原数组[0,1,2,3,4,5,6]

var newarr2=arr.concat([7,8]);
    console.log(newarr2); //返回的是[0,1,2,3,4,5,6,7,8]
    console.log(arr); //返回的依然是原数组[0,1,2,3,4,5,6]
var newarr3=arr.concat([7,8],[9,10])
    console.log(newarr3); //返回的是[0,1,2,3,4,5,6,7,8,9,10]
    console.log(arr); //返回的依然是原数组[0,1,2,3,4,5,6]
var newarr4=arr.concat([7,8,[9,10]]);
    console.log(newarr4); //返回的是[0,1,2,3,4,5,6,7,8,[9,10]]
    console.log(arr); //返回的依然是原数组[0,1,2,3,4,5,6]
//concat( ) 方法总结:
 //1.参数为非数组：直接在原数组后增加元素，并返回一个新数组
 //2.参数为数组：将这个数组参数与原数组拼起来并返回一个新数组
```

```
17  //3.参数为数组，并且这个数组参数内的元素还有数组，拼接后不会将数组参数内的数组变成非数组
```

- toString( )方法将数组直接转换为数组元素用逗号相连的字符串

```
1
2  var arr=[1,'2',[3,4],5];
3  var result=arr.toString( );
4      console.log(result); //结果为'1,2,3,4,5'
5      console.log(arr) //结果依然为原数组 [1,'2',[3,4],5]
```

- join( )方法将数组转换为字符串，参数为一个字符串，转换出来的字符串会以这个参数字符串作为每个数组之间的连接符

```
1  var arr = [1,2,3,4,5]
2  console.log(arr.join()); // 1,2,3,4,5 分隔符未填写，则默认为逗号“，”
3  console.log(arr.join("")); // 12345 分隔符为空字符
4  console.log(arr.join("-")); // 1-2-3-4-5
5  console.log(arr.join("|")); // 1|2|3|4|5
```

```
1
2  var resule=arr.join( '!!' );
3      console.log(result); //返回结果为    '1!!2!!3,4!!5'。可以看出，与toString( )方法得到的结果有所不同
4      console.log(arr); //结果依然为原数组 [1,'2',[3,4],5]
5   //不跟参数时，表示默认使用逗号作为连接符
6  var resule=arr.join( );
7      console.log(result);//结果为'1,2,3,4,5'
```

- sort( )方法排序

```
1  //数组排序sort( )方法会改变原数组
2  var arr=['5','1',4,2,'3']
3   //不跟参数时
4      arr.sort( );
5      console.log(arr); //返回: ["1", 2, "3", 4, "5"]
6   //不跟参数时，仅仅能够升序排序，并且，只会比较第一位数，比如18只会取1来比
7
8      //跟参数
```

```
var arr=[1,88,5,6,21,33,67];
    arr.sort(function (a,b){
 return a-b;        //升序排列
 //return b-a;   //降序排列
})
    console.log(arr); //返回[1, 5, 6, 21, 33, 67, 88]
```

- reverse( )方法进行数组翻转改变原数组；也可以检查一个字符串是否是回文，

```
//翻转
  var arr=[1,2,3,4];
    console.log(arr.reverse( )); //[4,3,2,1]
    console.log(arr); //[4,3,2,1]

//回文
 var str1 = '676';
     function frontEnd(str) {
        return str === Array.from(str).reverse().join('');
  }
     console.log(frontEnd(str1)); //true
     console.log(frontEnd('str123')); //false
     console.log(frontEnd('上海自来水来自海上')); //true
```

- every() 方法测试一个数组内的所有元素是否都能通过某个指定函数的测试。它返回一个布尔值

```
var arr = [3, 6, 9, 12, 31];
    console.log(arr.every(function (el) {
        return el % 3 === 0;
 })); //false
```

- some() 方法测试数组中是不是至少有1个元素通过了被提供的函数测试。它返回的是一个Boolean类型的值

```
 console.log(arr.some(function (el) {
      return el % 3 === 0;
 })); //true
```

- reduce() 方法对数组中的每个元素按序执行一个由您提供的 reducer 函数，每一次运行 reducer 会将先前元素的计算结果作为参数传入，最后将其结果汇总为单个返回值

语法:

```
array.reduce(function(previousValue, currentValue, currentIndex, arr), initialValue)
```

function接受四个参数:

- previousValue: 初始值（或者上一次回调函数的返回值）
- currentValue: 当前元素值
- currentIndex: 当前索引
- arr: 调用 reduce 的数组

回调函数第一次执行时，previousValue 和 currentValue 的取值有两种情况:
如果调用 reduce() 时提供了 initialValue，previousValue 取值则为 initialValue，currentValue 则取数组中的第一个值。
如果没有提供 initialValue，那么 previousValue 取数组中的第一个值，currentValue 取数组中的第二个值

```
var arr = [100, 200, 300];
var result = arr.reduce(function(previousValue, currentValue, currentIndex){
    console.log(previousValue, currentValue, currentIndex);
    return 87;
    //第1次打印：previousValue：0， currentValue：就是第1个元素， currentIndex：0，
    //第2次打印：previousValue：100， currentValue：就是第2个元素， currentIndex：1，
    //第3次打印：previousValue：100， currentValue：就是第3个元素， currentIndex：2，
 }, 0);
    console.log(result);
    //0 100 0
    //87 200 1
    //87 300 2
    //87
```

reduce应用:

```
<div id="app">
    我是{{name}}，今年{{age}}岁了，性别{{gender}}，
    我来自{{from.province}}省{{from.city}}市，
    {{from.ext.school}}大学{{from.ext.major}}专业
</div>
<script>
 var data = {
```

```
        name: 'ABC',
        age: 19,
        gender: 'female',
    from:{
        province:'贵州',
        city:'六盘水',
        ext:{
            school:'六盘水师范学院',
            major:'物联网工程'
            }
         }
     };
/** 拿到要操作的标签 */
var app = document.querySelector('#app');//拿到绑定模板里面的数据
/** 拿到标签里面的原始内容*/
var appHtml = app.innerHTML;
/** 拿到标签内容里面的所有的{{  }}里面的内容：属性*/
var appHtmlMustcahe = appHtml.split('{{');//模板格式是{{}}，所以用{{来分割
/** 使用forEach对数组进行循环*/
appHtmlMustcahe.forEach(function (item) {
//查找字符串(}})在另外一个字符串(item)里面第一次出现的位置

//变量是用来缓存数据的，
//如果一个数据只用一次，你完全可以不用定义一个变量来缓存这个数据，
//或者更准确的说，你不应该再去声明一个变量

var ind = item.indexOf('}}');//找到}}出现的位置，同时也可以判断当前元素是否包含}}
    if (ind > -1) { //-1表示不存在，>-1就表示存在了
        var k = item.substring(0, ind); //字符串截取，只要{{}}里面的内容
            console.log(k);
    //进行字符串替换,并重新赋值给原来的字符串
    //appHtml = appHtml.replace(`{{${k}}}`, data[k]); //k是变量，不能yong .  一层

    /*************************************************************************
    例子①：
          k:  'from.city'
          k.split('.') === >  ['from', 'city']
    第1次执行：previousValue:  data,  item: 'from'   previousValue[item]===> data['from']
    第2次执行：previousValue:  data['from'], item: 'city'
    previousValue[item]===>data['from']['city']
```

```
例子②：
      k:  'gender'
      k.split('.') === >  ['gender']
 第1次执行：previousValue：data，  item：'gender'   previousValue[item]===> data['gender']
 例子③：
      k:  'from.ext.school'
      k.split('.') === >  ['from', 'ext', 'school']
第1次执行：previousValue：data，  item：'from'   previousValue[item]===> data['from']
第2次执行：previousValue：data['from']，  item：'ext'   previousValue[item]===> data['from']['ext']
第3次执行：previousValue：data['from']['ext']，item：'school'  previousValue[item]===>data['from']['ext']['school']
*/

appHtml = appHtml.replace(`{{${k}}}`, k.split('.').reduce(function(previousValue, item){
    //字符串模板的使用和对象的属性值访问，k值是是变量，所以只能用[k]来获取data里面的对应的属性值
        return previousValue[item];
       }, data));
    }
});
//把处理之后的最新的内容显示到页面之上
   app.innerHTML = appHtml;
</script>
```

push()方法应用：

```
var a = ['A1', 'A2', 'B1', 'B2', 'C1', 'C2', 'D1', 'D2']
var b = ['A', 'B', 'C', 'D']
var newArr = [] // 建个空数组

for (var i = 0; i < a.length; i++) { // 把a数组push到新数组中
     newArr.push(a[i])
    if (i > 0 && (i + 1) % 2 == 0) { // 当数组索引大于0，且为2的倍数，把b数组的元素push到新数组
          newArr.push(b[((i + 1) / 2) - 1])
       }
   }
```

```
    console.log(newArr);//['A1', 'A2', 'A', 'B1', 'B2', 'B', 'C1', 'C2', 'C', 'D1', 'D2', 'D']
```

this指向应用：

```
function obj(name) {
     console.log(this);
   if (this === window) { //判断通过何种方式调用
        if (name) {
            this.name = name;
         } else {
             this.name = 'name1';
           }
         return this;
       }
  }
     obj.prototype.name = "name2";
//如果通过函数方式调用,this会指向window
var a = obj("name1");

//如果通过new方式调用,this会指向实例化后的对象,obj()
var b = new obj();

//如果 函数调用的时候不带参数,默认name为name1
var c = obj();
    console.log(a.name);//name1
    console.log(b.name);//name2
    console.log(c.name);//name1
```

forEach、some、filter的区别：

```
<script>
        var arr = ['red', 'green', 'blue', 'pink'];
        // 1. forEach迭代 遍历
        arr.forEach(function (value) {
            if (value == 'green') {
                console.log('用forEach找到了该元素');
                return true; // 在forEach 里面 return 不会终止迭代
```

```
            }
            console.log(11);
        })

        console.log('-----------------------------------------');


        //2 如果查询数组中唯一的元素，用some方法更合适，
        arr.some(function (value) {
            if (value == 'green') {
                console.log('用some找到了该元素');
                return true; //  在some 里面 遇到 return true 就是终止遍历 迭代效率更高
            }
            console.log(11);
        });

        console.log('-----------------------------------------');

        //3 filter过滤
        arr.filter(function (value) {
            if (value == 'green') {
                console.log('用filter找到了该元素');
                return true; // filter 里面 return 不会终止迭代
            }
            console.log(11);
        });
    </script>
```